Naval Ocean Research and Development Activity
February 1988 Report 203

Implementation Issues for Level 0 Image Processor Software within an Application Package

James E. Lennox
Mapping, Charting, and Geodesy Division
Ocean Science Directorate

Foreword

The Mapping, Charting, and Geodesy Division has been tasked by the Defense Mapping Agency to develop methods for extracting bathymetry information from multimode sensor data. This task requires the use of sophisticated image processors and software packages. This paper describes the work done to accomplish interfaces to two different image processors and a methodology for the general case of implementing image processors to a high-level software package.

W. B. Moseley

W. B. Moseley
Technical Director

A. C. Esau

A. C. Esau, Captain, USN
Commanding Officer

Executive Summary

Image processing is rapidly emerging as a field with many interesting areas for the computer scientist. Packages are available that allow images to be manipulated in an interactive environment by applying functions to the image. Functions are defined as transformations from the image memory domain to the display domain, which may permanently alter image memory values. This paper describes minimum hardware constraints and the scope of modifications necessary to implement different image processors within the Earth Resources Laboratory Applications Software (ELAS) environment. It also contrasts the impact on software engineering principles related to the implementation of low-level software to support the imaging functions.

Acknowledgments

This study would not have been possible without the guidance, encouragement, and suggestions provided by the government employees in the Pattern Analysis Branch of the Mapping, Charting and Geodesy Division. The author is especially indebted to Dr. Charles Walker of NORDA for his vision, patience and scientific leadership, and NASA's Earth Resources Lab for developing the ELAS image processing software.

This study was sponsored by the Oceanographer of the Navy (OP-096). The work was funded by Program Element 63704N, coordinated by Ms. Mary Clawson.

Contents

Implementation Issues for Level 0 Image Processor Software within an Application Package

Background

ELAS is an image manipulation package originally developed by the Earth Resources Laboratory (ERL). The package is comprised of over 600 modules. Modules are written in FORTRAN 4 and the host assembler language. The Pattern Analysis Laboratory (PAL) of the Naval Ocean Research and Development Activity (NORDA) needed to secure ELAS to support research in the determination of bathymetry from Landsat and Thematic Mapper data. The host machine was a VAX 11/780 with a Gould IP 8500 image processor. Subsequently another version of ELAS was developed on a VAX 11/750 with a Grinnell 275 image processor. This report discusses the implementation of both systems through the development cycle and related software engineering issues. It is a comparison of hardware and low-level interfaces between the two machines, and a description of that impact on the software engineering-related issues.

Theory of Operation

ELAS is comprised of two types of routines: modules and callable subroutines. Modules are scheduled by the currently running process and the state is saved in a system of subfiles. Scheduling results from typing in the name of a major module as the next command to process.

Within the context of I/O in the ELAS image-manipulation system, COMD is the common image display module and LABL is the textual graphics and vector drawing module. Each of these modules interface with a low-level, callable subroutine, CIO, that provides a common I/O interface to the image processor.

Software Scope

The bulk of the programming effort provided subroutines CIO, ZOOM, RDSTAT, and TRKINT, which were not present and which were written in FORTRAN 77. This necessitated changes to modules ELAS, COMD, LABL, and FMGR. Changes were also necessary in subroutines RT, CURDIF, DISINT, FUNMEM, and FUNM2, GWRIT2.

Modules ELAS, FMGR, and RT were originally written for a 512 × 256 pixel image processor with fewer image and graphics planes. Changes in these modules involved defining the number and size of image planes and the number of peripherals. The driver name and setup codes also had to be changed to reflect the particular image processor in use.

Modules COMD and LABL were modified to allow zoom and scroll capability and to allow hardware to clear image and graphics planes, which increased the speed of these operations considerably. The build table section was updated to allow color table values from 0 to 255 instead of the 0 to 15 range previously implemented. The print color table section was enhanced to print values from 0 to 255 rather than 0 to 15. Finally, these modules were changed to use a trackball interface for determining cursor position rather than ASCII characters from the VT100 keyboard.

Subroutines CURDIF, DISINT, FUNMEM, and FUNM2 were also changed to use the trackball interface instead of ASCII sequences from the terminal.

Subroutine GWRIT2 was modified to pass 4-byte parameters to ILBIT to allow correct subroutine linkage. Previously implemented versions of GWRIT2 were developed on older PDP-11 machines which used a 16-bit word rather than the standard 32-bit word used on the VAX architecture. Problems in this area manifest themselves in a unique way because the VAX architecture stores its most significant bits in the lowest address. Thus a valid integer passed to an ILBIT became a zero or a large negative number in that routine's formal parameter list. The value erroneously passed depends on the value of the integer in the calling routine. This change corrected a problem encountered with writing graphics in the vertical direction.

An additional software change was a modification to GRDRV, the DR11-B driver. The driver was modified to provide compatibility with the VMS 4.2 operating system. $DYNDEF was added to the driver to map in the dynamic definitions that were resident in a different library in the VMS 3.6 software.

Hardware Scope

The existing hardware configuration did not have sufficient lookup tables to simultaneously map an intensity scale function and a color lookup function. This is because the only lookup tables available were on the image function memory board and only one function could be loaded at a time. Another set of functions was available on the image processor card, but the output of these tables is routed back to the image planes, permanently altering their values. This was considered an unacceptable approach during true color operation because three image planes are used to hold the RGB spectrum and three scratch planes would also be needed. The Grinnell has only five image planes.

The cleanest solution and the one implemented was to change the image video driver board to a function video driver board. The boards operate identically but the function video driver board provides another set of lookup tables to the system. There was a problem associated with the implementation: slot #33 did not have read-back lines or control signals for the lookup tables. This problem was solved by modifying the back plane to accommodate those signals to the slot. The system now provides two sets of lookup tables to the system in a sequential path from image memory to the video drivers. Appendix A documents the changes necessary to reconfigure the Grinnell image processor for the function video driver board.

The trackball interface provided an additional set of problems. There is a hardware-design-related problem with polling the trackball interrupt registers. The enter push-button switch must be depressed before any other switches can be sensed in the interface register. Two resistor packs were used in the trackball, -1 and -3 types. They both have 16 pins but the -3 has 8 resistors in the bridge instead of 15. This causes a timing problem when reading the trackball interrupt register, specifically the enter button. Consequently the interface loses synchronization with the system-provided QIO and a successful read does not take place. The problem is then complicated by the fact that QIO thinks it got a successful read, as signified by the IO-status block word one, and the number of bytes read is signified by word two. This means that the system service routine $SYNCH cannot be used to indicate either condition. The problem did not respond to the use of event flags since the QIO routine believes it is getting a valid read. Slowing the loop down by initiating a $WAITFR (or a do-nothing loop) eliminates the problem and does not seem to affect the use of the trackball with ELAS. One is primarily interested in the state at that time and not in obtaining an interrupt. Slowing down the loop also precluded multiple reads in a loop as an approach with a limit on the number of reads. The delay does not adversely impact performance. The resolution of this problem was very time and labor intensive.

Software Engineering Issues

A variety of lessons can be learned from the implementation of ELAS in an environment with a specific image processor. The hardware and software support offered by a vendor may dictate design issues so that a clean implementation cannot be achieved. This can be experienced with other image-processing packages as well.

Vendor software is a major issue in the decision to buy a particular piece of hardware. However, it may be preferable to implement your own software solutions because of the level of complexity that may be added to the software package. Complexity is determined by communication structures used by designers of the user package and the image-processing vendor's library.

In modern software packages, communication between different independent routines is achieved by message passing schemes. These are usually global or common variables, depending on the language choice, and are true of both interactive and library callable subroutines. To use the vendor-supplied routines, the message system must be established by the application software package, which creates a structure on top of the user package, which obviously fails to localize the functions of the image processor with relationship to the other software in the user package. This method is wrong from a software engineering viewpoint for two reasons: complexity and control of data flow.

Complexity is increased by many factors that present at least two alternatives. First, the software maintainer must know in an intimate way two different user packages. Thus, the programmer must know the image-processing system even if the primary interest is in the development of statistical manipulation capabilities for the system. The other alternative is to have two different people responsible for maintenance of the system, neither of whom is fully skilled to handle problems in other parts of the software. The second alternative impedes the speed of problem determination and may have great impact in a production environment. Also, there could be more overhead in communication between programmers assigned to modify or maintain the software, including a lengthened learning curve.

Good software practice demands that a module receive information on a need-to-know basis. At the very least, write-access to a variable should be limited and regulated by a good design methodology. Adding a layer of variables for message-passing schemes places the entire structure in a wide-open environment. The latter may cause a variety of problems, including distortion of the message if the variable's state is compromised, and in a large system it could be extremely difficult to locate.

Further, placing a layer on top of the applications package completely destroys portability, since all modules would have to be changed to accommodate

a new vendor's message-passing protocol. It must also be pointed out that vendor software is generally proprietary and may be sold separately and not included with the hardware. Portability of the package to another lab may be impossible or more expensive; therefore, display routines should be isolated and the vendor's library should be used only when the goal of portability is not compromised.

The image-processing system can be likened to a layer of abstractions from the software interface viewpoint. The Gould IP8500 image processor provides a level 0 function IP8Q, which is a pseudo-driver on top of the VAX-supplied QIO routine. The parameters passed to IP8Q include mnemonics that tell IP8Q what function to perform on behalf of the user's application. It may take several calls to affect a principal function of the image processor. The system also provides parameters for selecting the register and the board to modify where a polyboard or polyregister environment exists. Thus, modification to a particular register on a particular board causes a unique function of the imaging system to occur (i.e., reading an image into image memory). The IP8Q is performing the task of packaging bit patterns that the machine can understand and handing them to QIO. In this manner the mnemonics generate streams of bit patterns which, packaged with other parameters, instruct the machine to perform a particular function. Grouping these calls together formulates level 1 calls, which are a level of abstraction higher than level 0 calls.

The Grinnell GMR275 does not provide an abstraction at this level. Therefore, it was necessary to form an array of bit patterns and pass the array to QIO. Mnemonics for each primitive instruction on the boards are associated with a hexidecimal code representing that instruction's opcode. Specific instructions are formed by logically "and-ing" or logically "or-ing" a parameter with an instruction mask. In this vein, streams of instructions are generated to effect a major component of the imaging engine. Thus, functionality of instruction packets were made isomorphic to the functions of the Gould IP8500 though individual instructions were not isomorphic.

To modularize the functionality of the common I/O interface (CIO) for the Grinnell, a strategy of localization was adopted. Initialization during each major function was performed within that stream of instructions, which made each function of CIO completely contained within that portion of the code rather than relying on initialization coming into the routine at the top. This allowed an extremely crisp case structure to be implemented with a computed GOTO, and, together with the use of mnemonics, made the development quick and easy since all logic problems were isolated within a small section of code within the routine.

Conclusions

A wide variety of image processors are on the market ranging in price from a few thousand to several hundred thousand dollars. The higher-priced systems provide hardware-implemented solutions to software-related problems such as histogram equalization or warping and stretching. In order for the image-manipulation packages to remain independent they must continue to provide these functions in software. This lessens the risk of becoming a hardware-dependent package. While not machine specific, ELAS is architecture specific, treating the image processor as just a frame buffer. This attitude should be relaxed to begin taking advantage of the tremendous advances in today's machines. However, most applications in the image-processing domain do not require instantaneous results so the lack of hardware functions does not significantly degrade the capacity to use the package. It is recommended that these advanced features be viewed as trade-offs in terms of throughput. Cheaper hardware costs allow more workstations, which may provide more throughput than a single machine running at the state of the art. However a minimum configuration should include the following:

1. A single video output controller for each workstation.

2. Four 512 × 512 by eight-bits-deep image planes for each work station (three planes configured for RGB and one plane for graphics).

3. Two sets of lookup tables, one for functions and one for color mapping, available to each workstation. The tables should each be 256 × 8 bits deep. The color lookup table should contain three 256 × 8-bits-deep tables.

4. One trackball interface with multiple interrupt buttons.

5. A monitor capable of displaying 512 × 512 by eight bits deep at one time. However most packages are moving toward a 1024 × 1024 by twelve-bits-deep display.

Beyond these considerations, speed versus price becomes an issue. The time required to classify an image with the iterative maximum likelihood estimators grows nonlinearly. Thus an increase in the display size will cause computational time to grow by more than a factor of two.

It may become economical to include such features as an array processor board or a histogram equalization board. Image windowing can also come in handy when large files are to be zoomed and scrolled.

The standard configuration of a vendor's image processor should be carefully scrutinized. They do not all provide standard hardware features. The Grinnell GMR 275 image processor is a prime example. It provided only one set of lookup tables, which made a

backplane modification and a board purchase necessary. The type of output also should be checked. Vendors usually supply RS-170A, 30-Hz interleaved, or 60 Hz noninterleaved. Some vendors supply a combination of RS-170 and one of the other two.

Turning to software engineering issues, portability cannot be maintained if the vendor's package is placed on top of the existing applications package. This would limit the user to systems with similar hardware. It was also pointed out earlier in this paper that placing the vendor's package on top of the existing imaging package unnecessarily adds to complexity; therefore, it should be avoided. Avoidance will require developing level 0 software by the support group for the user application package. Price being equal, the software with the highest level of abstraction in its level 0 calls should be chosen, since it will minimize the development cycle.

Regarding the development of the common I/O interface, the assignment of opcodes to mnemonic names increased readability, because they are more descriptive than hexidecimal codes. Additionally, there was less error due to misdefining the opcodes since, once defined properly, they became a nonissue in the development cycle. Quintessentially, the most significant contributing factor was the complete isolation of functions within the routine. There were a number of reasons. First, the area of interest was localized to a block of code rather than splitting it by initializing at the top section. This concentrates functionality within the block of code making comprehension much easier. Second, because all initialization is performed within the block, the effects of extraneous values left in a register are minimized. This provides a complete implementation of the functional subcomponents in each block of code. Understandability was partitioned or chunked by the isolation, which implies that only that subcomponent of interest need be grasped rather than an entire routine. The image processors in general are faster than the VAX 11/780. Thus duplication of the initialization, which amounted to a few additional lines of code, is negligible because initialization at the top requires FORTRAN conditional logic, which is slower than executing three or four additional inline instructions in the image processor.

Appendix A: Wiring Modifications to the Grinnell GMR 275 Image Processor

The backplane wiring list can be read as follows. Each connector has two rows which we will call bottom and top. The bottom row is designated PS while the top row is designated CR.

Each row has two connectors which we will call left and right. The left connector is designated by the numeral 1 while the right connector is designated by the numeral 2.

On each connector the pins are labeled from 1 to 50 starting on the bottom row.

The first two numbers represent the slot number for the board.

For example: 331PS34

33 → The thirty-third slot on the backplane;

1 → The left side connector;

P S→ The bottom row side;

3 4→ The thirty-fourth pin.

The modifications are marked on the five pages which follow. The numbers 9XDB14 through 9XDB00 are the designations for the lookup table address lines.

BACKPLANE WIRELIST NO. 102918-E Page 33 of 66

9XRBDR

03	1	P	S	25		
05	2	C	R	41		
07	1	P	S	41		
08	↑	↑	↑	↑	↑	↑
09						
10						
11						
27						
29	↓	↓	↓	↓	↓	↓
33	1	P	S	41		
31	1	P	S	41		
35	1	P	S	41		
01	2	P	S	39		

9TRBAK

03	1	P	S	38		
05	2	C	R	42		
07	1	C	R	41		
08	↑	↑	↑	↑	↑	↑
09						
10						
11						
27						
29	↓	↓	↓	↓	↓	↓
33	1	C	R	41		
31	1	C	R	41		
35	1	C	R	41		

9EEWA0

05	2	C	R	38		
13	1	P	S	20		
14	↑	↑	↑	↑	↑	↑
15						
16						
17						
18						
19						
20						
21						
22	↓	↓	↓	↓	↓	↓
23	1	P	S	20		

9EEWA1

05	2	C	R	37		
13	1	P	S	21		
14	↑	↑	↑	↑	↑	↑
15						
16						
17						
18						
19						
20						
21						
22	↓	↓	↓	↓	↓	↓
23	1	P	S	21		

9EEWA2

05	2	C	R	36		
13	1	P	S	22		
14	↑	↑	↑	↑	↑	↑
15						
16						
17						
18						
19						
20						
21						
22	↓	↓	↓	↓	↓	↓
23	1	P	S	22		

9XDB15

03	1	C	R	40		
07	1	P	S	40		
08	↑	↑	↑	↑	↑	↑
09						
10	↓	↓	↓	↓	↓	↓
11	1	P	S	40		
12	2	C	R	33		
29	1	P	S	40		

T22

BACKPLANE WIRELIST NO. 102918-E Page 34 of 166

9XDB14

03	1	C	R	39		
07	1	C	R	40		
08	↑	↑	↑	↑	↑	↑
09						
10	↓	↓	↓	↓	↓	↓
11	1	C	R	40		
12	2	C	R	32		
29	1	C	R	40		

9XDB13

03	1	C	R	38		
07	1	P	S	39		
08	↑	↑	↑	↑	↑	↑
09						
10	↓	↓	↓	↓	↓	↓
11	1	P	S	39		
12	2	C	R	31		
29	1	P	S	39		

9XDB12

03	1	C	R	37		
07	1	C	R	39		
08	↑	↑	↑	↑	↑	↑
09						
10	↓	↓	↓	↓	↓	↓
11	1	C	R	39		
12	2	C	R	30		
29	1	C	R	39		

9XDB11

03	1	C	R	36		
07	1	P	S	38		
08	↑	↑	↑	↑	↑	↑
09						
10	↓	↓	↓	↓	↓	↓
11	1	P	S	38		
13	1	C	R	17		
31	1	C	R	19		
29	1	P	S	38		
33	1	P	S	38		
12	2	C	R	29		

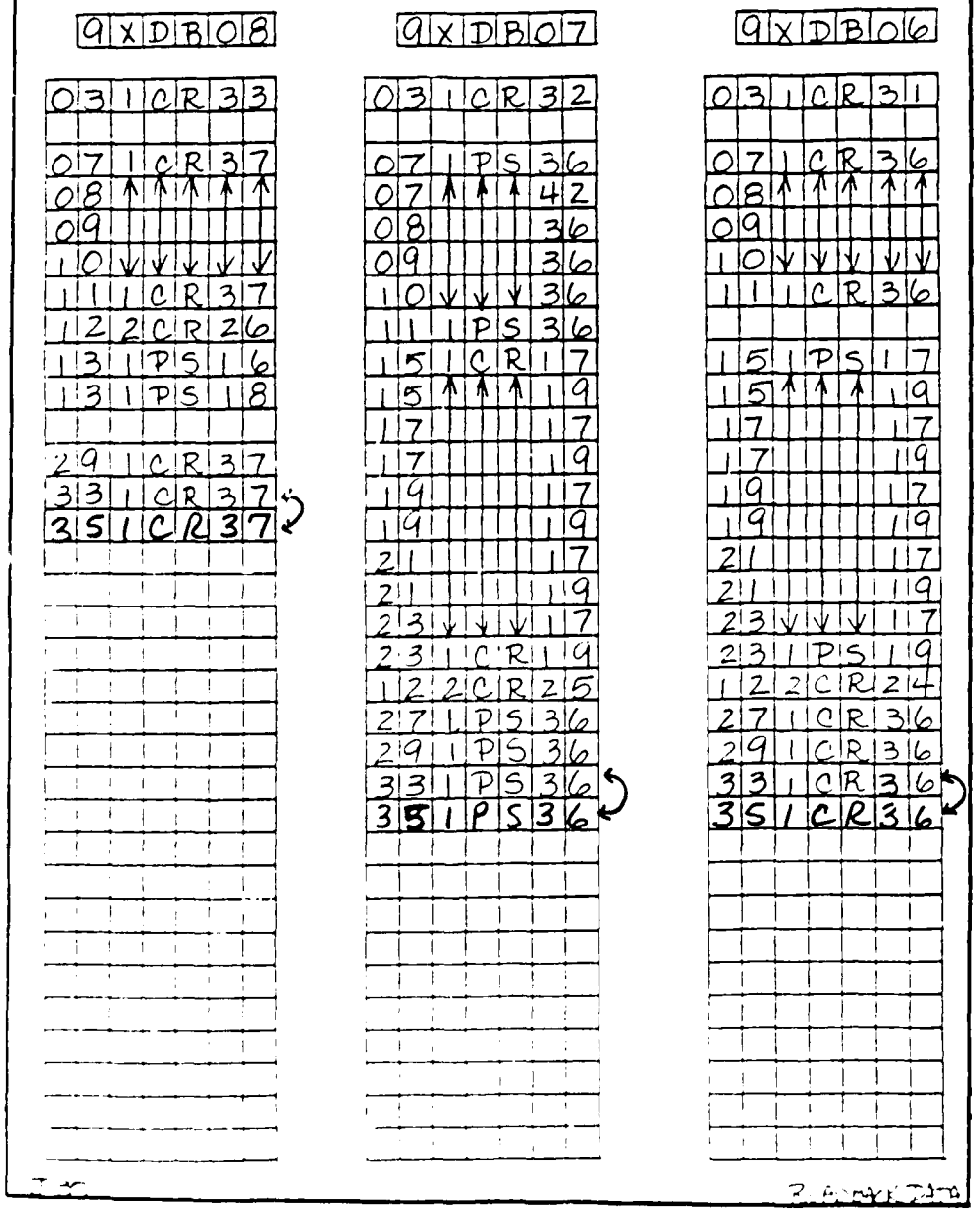
9XDB10

03	1	C	R	35		
07	1	C	R	38		
08	↑	↑	↑	↑	↑	↑
09						
10	↓	↓	↓	↓	↓	↓
11	1	C	R	38		
13	1	P	S	17		
13	1	P	S	19		
29	1	C	R	38		
33	1	C	R	38		
12	2	C	R	28		

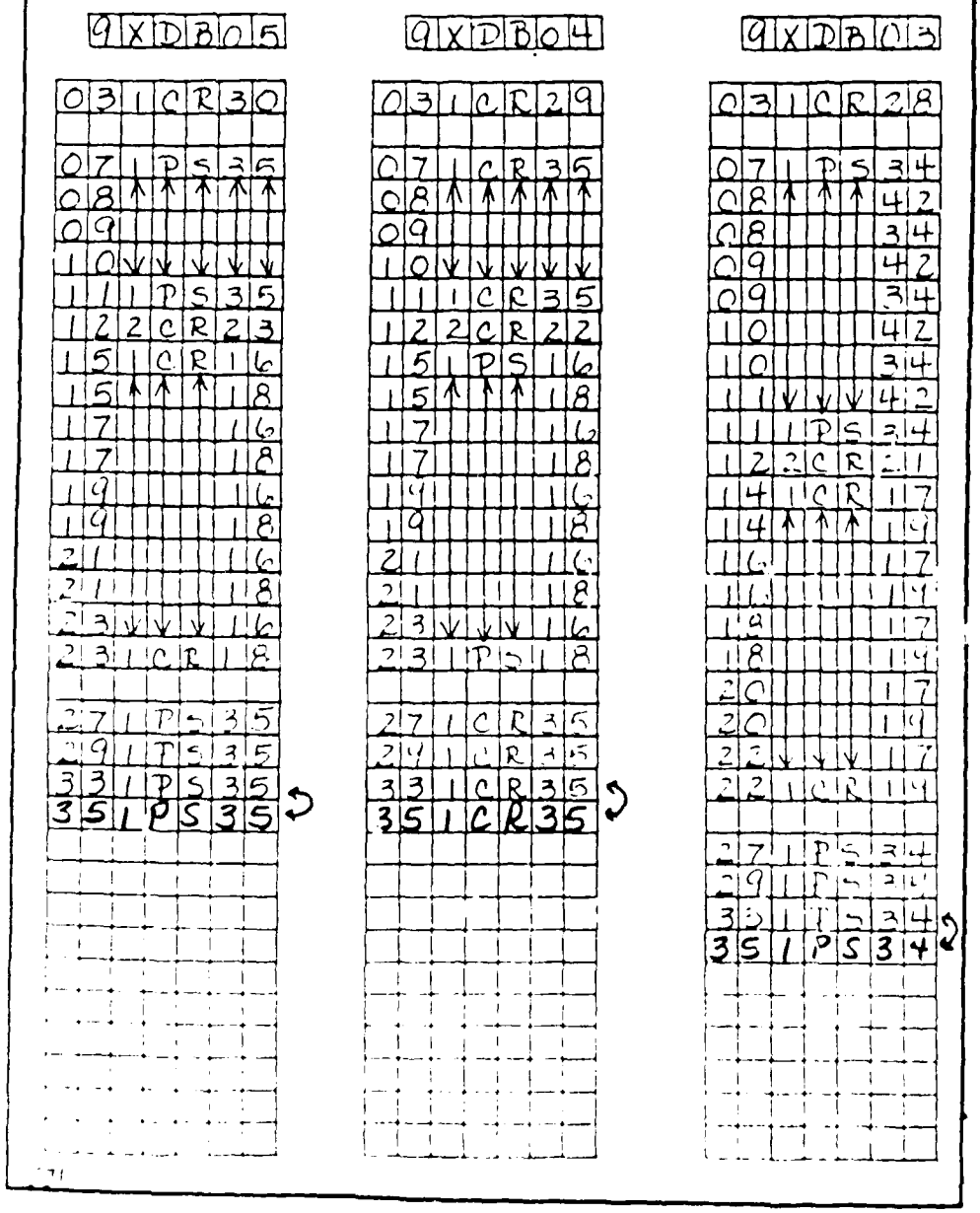
9XDB09

03	1	C	R	34		
07	1	P	S	37		
08	↑	↑	↑	↑	↑	↑
09						
10	↓	↓	↓	↓	↓	↓
11	1	P	S	37		
13	1	C	R	16		
13	1	C	R	18		
29	1	P	S	37		
33	1	P	S	37		
12	2	C	R	27		

BACKPLANE WIRELIST NO. 102918-E Page 35 of 66



BACKPLANE WIRELIST NO. 102918-E Page 36 of 66



BACKPLANE WIRELIST NO. 102918-E Page 37 of 66

9XDB02	9XDB01	9XDB00
03 1 CR 27	03 1 CR 26	03 1 CR 25
07 1 CR 34	07 1 PS 33	07 1 CR 33
08 ↑ ↑ ↑ ↑ ↑	08 ↑ ↑ ↑ ↑ ↑	08 ↑ ↑ ↑ ↑ ↑
09	09	09
10 ↓ ↓ ↓ ↓ ↓	10 ↓ ↓ ↓ ↓ ↓	10 ↓ ↓ ↓ ↓ ↓
11 1 CR 34	11 1 PS 33	11 1 CR 33
12 2 CR 20	12 2 CR 19	12 2 CR 18
14 1 PS 17	14 1 CR 16	14 1 PS 16
14 ↑ ↑ ↑ 19	14 ↑ ↑ ↑ 18	14 ↑ ↑ ↑ 18
16 17	16 16	16 16
16 19	16 18	16 18
18 17	18 16	18 16
18 19	18 18	18 18
20 17	20 16	20 16
20 19	20 18	20 18
22 ↓ ↓ ↓ 17	22 ↓ ↓ ↓ 16	22 ↓ ↓ ↓ 16
22 1 PS 19	22 1 CR 18	22 1 PS 18
27 1 CR 34	27 1 PS 33	27 1 CR 33
29 1 CR 34	29 1 PS 33	29 1 CR 33
33 1 CR 34 ↺	33 1 PS 33 ↺	33 1 CR 33
35 1 CR 34 ↺	35 1 PS 33	35 1 PS 42 ↺
		35 1 CR 33 ↺

T32.

Appendix B: ELAS Routines Affected

```
©PY©-
©IMH0150008000©IMV0055010000©ISTF01©IS204©IC000©IT01000©IOP
C
C       CIO.FOR   FOR THE GRINNELL IMAGE DISPLAY
C
C       CALL CIO ( LU, IOP, NC, LINE, NBTS, BUFF, L )
C
C       LU    = LOGICAL UNIT ( NOT USED )
C
C       IOP   = COMTAL OPERATION CODE
C             = 1  FOR IMAGE READ
C             = 2  FOR IMAGE WRITE
C             = 3  FOR GRAPHIC READ
C             = 4  FOR GRAPHIC WRITE
C             = 5  FOR FUNCTION READ
C             = 6  FOR FUNCTION WRITE
C             = 7  FOR COLOR TABLE READ
C             = 8  FOR COLOR TABLE WRITE
C             = 9  FOR TARGET READ ( DISABLED )
C             = 10 FOR TARGET WRITE
C
C       NC    = COMPONENT NUMBER
C             = IMAGE / GRAPHIC NUMBER I IOP IS 1,2,3 OR 4
C               ALSO IF NC IS NEGATIVE, THE SCROLL REGISTER WILL BE
C               UPDATED AFTER THE OPERATION TO ALLOW THE IMAGE TO BE
C               ROLLED.
C             = UNUSED FOR IOP GREATER THAN 4
C
C       LINE = LINE NUMBER FOR IMAGE OR GRAHIC READ OR WRITE.
C              THIS NUMBER IS USED AS ELAS EXPECTS. I.E. THE
C              TOP LINE IS LINE 0.
C
C       NBTS = NUMBER OF BYTES ( NOT USED )
C
C       BUFF = DATA ARRAY
C
C       L     = STATUS ; NORMALLY EQUAL TO NBTS ; NEGATIVE ON ERROR
C
      SUBROUTINE CIO ( LU, IOP, NC, LINE, NBTS, BUFF, L )
      IMPLICIT INTEGER*4 ( A - Z )
      INTEGER*2 LN, ELM
      INTEGER*2 GR, IMG, MASK, COL, BUFF(512), B(600), IOSB(4),
     *          WID, LSM, WGD, WAC, LWM, LUM,
     *          ERS, ERL, SLU, EGW, LER, LEA,
     *          LDC, NOP, LPR, LPR1, LPR2, LPR3,
     *          LPR4, LPR5, SPD, LPA, LPD, RPD,
     *          BIT15
C
C     THE FOLLOWING PARAMETERS DEFINE THE GRINNEL OPCODES SYMBOLICLY.
C
      PARAMETER
     *(WID='0000'X,  LSM='1000'X,  WGD='2000'X,
     * WAC='2400'X,  LWM='2800'X,  LUM='2C00'X,
     * ERS='3000'X,  ERL='3400'X,  SLU='3800'X,
```

```
     * EGW='3C00'X,  LER='4000'X,  LEA='4800'X,
     * LEB='5000'X,  LEC='5800'X,  LLR='6000'X,
     * LLA='6800'X,  LLB='7000'X,  LLC='7800'X,
     * LDC='8000'X,  NOP='9000'X,  LPR='C000'X,
     * LPR1='C200'X, LPR2='C400'X, LPR3='C600'X,
     * LPR4='C800'X, SPD='A000'X,  LPA='B000'X,
     * LPD='D000'X,  RPD='E000'X,
C
C    BIT MASKS
C
     * BIT15='8000'X)
C
C     CB IS AN ARRAY USED FOR CFSUB CALLS
C
      INTEGER*4 CB(4)
C
C     FIRST IS USED TO TELL CIO WHEN IT IS FIRST CALLED.
C
      DATA FIRST /0/
      DATA CB / 4H1FCT,0,0,0/
      DATA ICH/1/
      EXTERNAL IO$_READLBLK
      EXTERNAL IO$_WRITELBLK
C
C     IF THIS IS THE FIRST TIME INTO CIO, OPEN THE GRINNEL
C
      IF ( FIRST.EQ.0 ) THEN
        L = SYS$ASSIGN ( 'GRA0:',GR_LU,,)
        IF ( L.NE.1 ) THEN
          WRITE ( 6,1) L
 1        FORMAT(' UNABLE TO ASSIGN THE GRINNELL; STATUS = ',Z8 )
        ELSE
          FIRST = 1
        END IF
      END IF
C
C
C
      GO TO (100, 200, 300, 400, 500, 600, 700, 800, 900,
     *       1000, 1100, 1200, 1300), IOP
C
C
C     READ IMAGE
C
  100 CONTINUE
        LN = 511 - LINE                    ! REVERSE ORDER TO MATCH ELAS
        IMG  = ABS (NC)
        IMG  = 2 ** (IMG - 1)
        B(1) = IOR ( LWM, '0020'X )
        B(2) = IOR ( LUM, '0002'X )
        B(3) = IOR ( LLA, LN      )
        B(4) = IOR ( LEA, '0001'X )
        B(5) = IOR ( LEB, '0001'X )
        B(6) = IOR ( LDC, IMG     )
```

```
        B(7) = IOR ( LSM, '00FF'X )
        B(8) = IOR ( SPD, '0100'X )
        B(9) = IOR ( RPD, '0000'X )
        STAT = SYS$QIOW ( , %VAL( GR_LU ), IO$_WRITELBLK ,
     *          IOSB,,,B, %VAL( 18 ),,,,)
        STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_READLBLK,
     *          IOSB,,,B, %VAL ( 1024 ),,,,)
        CALL CIOPCK ( B, BUFF, 512 )
        L = 512
      RETURN
C
C     WRITE IMAGE
C
  200 CONTINUE
        LN = 511 - LINE                    ! REVERSE ORDER TO MATCH ELAS
        IMG  = ABS (NC)
        IMG  = 2 ** (IMG - 1)
        B(1) = IOR ( LWM, '0000'X )
        B(2) = IOR ( LUM, '0002'X )
        B(3) = IOR ( LLA, LN       )
        B(4) = IOR ( LEA, '0000'X )
        B(5) = IOR ( LEB, '0001'X )
        B(6) = IOR ( LDC, IMG      )
        B(7) = IOR ( LSM, '00FF'X )
        B(8) = IOR ( SPD, '0200'X )
        B(9) = IOR ( LPR, '0200'X )
        DO I = 1, 512
          B(I+9) = BUFF(I)
        END DO
        NTW = 512 + 18
        STAT = SYS$QIOW ( , %VAL( GR_LU ), IO$_WRITELBLK ,
     *          IOSB,,,B,%VAL( NTW ),,,,)
        L = 512
      RETURN
C
C     READ GRAPHICS
C
  300 CONTINUE
        GR = ABS ( NC ) / 2
        GR = 2**(GR + 8)
        LN = 511 - LINE                    ! REVERSE ORDER TO MATCH ELAS
        B(1)  = IOR ( SPD,  '0001'X )
        B(2)  = IOR ( LPR1, '00C0'X )
        B(3)  = IOR ( LPR2, '00A0'X )
        B(4)  = IOR ( LPR3, '0010'X )
        B(5)  = IOR ( LDC,  GR      )
        B(6)  = IOR ( LSM,  '0F00'X )
        B(7)  = IOR ( LEA,  '0001'X )
        B(8)  = IOR ( LLA,  LN      )
        B(9)  = IOR ( LEB,  '0001'X )
        B(10) = IOR ( LLB,  '0000'X )
        B(11) = IOR ( LUM,  '0002'X )
        B(12) = IOR ( LWM,  '0840'X )
        B(13) = IOR ( SPD,  '0100'X )
```

```
      B(14) = IOR ( RPD,  '0C00'X )
      STAT = SYS$QIOW ( , %VAL( GR_LU ), IO$_WRITELBLK ,
     *        IOSB,,,B, %VAL( 28 ),,,,)
      STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_READLBLK,
     *        IOSB,,,B, %VAL ( 512 ),,,,)
      CALL CIOGPK ( B, BUFF, 512 )
      CALL SWL ( BUFF, 64 )
      L = 64
     RETURN
C
C     WRITE GRAPHICS
C
  400 CONTINUE
      CALL SWL ( BUFF, 64 )
      GR = ABS ( NC ) / 2
      GR = 2**(GR + 8)
      LN = 511 - LINE                    ! REVERSE ORDER TO MATCH ELAS
      B(1)  = IOR ( SPD, '0001'X )
      B(2)  = IOR ( LPR1,'00CF'X )
      B(3)  = IOR ( LPR2,'00AF'X )
      B(4)  = IOR ( LPR3,'001F'X )
      B(5)  = IOR ( LDC, GR      )
      B(6)  = IOR ( LSM, '0F00'X )
      B(7)  = IOR ( LEA, '0000'X )
      B(8)  = IOR ( LLA, LN      )
      B(9)  = IOR ( LEB, '0008'X )
      B(10) = IOR ( LLB, '0000'X )
      B(11) = IOR ( LUM, '0002'X )
      B(12) = IOR ( LWM, '0840'X )
      B(13) = IOR ( SPD, '0200'X )
      B(14) = IOR ( LPR, '0820'X )
      DO I = 1, 32
         B(I+14) = BUFF(I)
      END DO
      NTW = 64 + 14
      NTW = NTW * 2
      STAT = SYS$QIOW ( , %VAL( GR_LU ), IO$_WRITELBLK ,
     *        IOSB,,,B, %VAL ( NTW ),,,,)
      CALL SWL(BUFF,64)
      L = 64
     RETURN
C
C     READ FUNCTION
C     SINCE THE GRINNELL HAS 3 LOOKUP TABLES WHICH HAVE BEEN
C     IMPLEMENTED TO SERVE AS BOTH FUNCTION & COLOR TABLE,
C     THE READ FUNCTION IS CODED TO READ FROM A SUBFILE.
C
  500 CONTINUE
      CB(3) = 1
      CB(4) = 128
      CALL CFSUB ( 5, CB, BUFF )
      L = 512
     RETURN
C
```

```
C     WRITE FUNCTION
C
  600 CONTINUE
        CB(3) = 1
        CB(4) = 128
        CALL CFSUB ( 6, CB, BUFF )
        B(1) = IOR ( SPD,  '0002'X )
        B(2) = IOR ( LPA,  '0C00'X )
        DO I = 1, 256
           B(I+2) = IOR ( LPD, BUFF(I))
        END DO
        NTW = 512 + 4
        STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_WRITELBLK ,
     *          IOSB,,,B, %VAL( NTW ),,,,)
        L = 512
      RETURN
C
C     READ COLOR TABLE
C     THIS OPTION READS FROM A SUBFILE RATHER THAN THE GRINNELL.
C
  700 CONTINUE
        CB(3) = 129
        CB(4) = 256
        CALL CFSUB ( 5, CB, BUFF )
        L = 512
      RETURN
C
C     WRITE COLOR TABLE
C
  800 CONTINUE
        CB(3) = 129
        CB(4) = 256
        CALL CFSUB ( 6, CB, BUFF )
        B(1) = IOR ( SPD, '0001'X )
        DO ICOL = 1, 3
          MASK = ( ICOL - 2 ) * '800'X
          IF ( MASK.LT.0 ) MASK = '400'X
          B(2) = IOR ( LPA, MASK )
          MASK = 15 * 16**( ICOL - 1 )
          DIV = 16**( ICOL - 1 )
          DO LOC = 1, 256
            COL = IAND ( BUFF ( LOC ), MASK )
            COL = COL / DIV * 17
            B ( LOC + 2 ) = IOR ( LPD, COL )
          END DO
          NTW = 512 + 4
          STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_WRITELBLK,
     *            IOSB,,,B, %VAL( NTW ),,,,)
        END DO
        L = 512
      RETURN
C
C     READ TARGET ONLY ONE CURSOR OF FOUR ACTIVATED
C
```

```
  900 CONTINUE
        B(1) = IOR ( SPD, '0080'X )
        B(2) = IOR ( LPR, '0011'X )
        B(3) = IOR ( LPA, '0000'X )
        B(4) = IOR ( RPD, '0000'X )
        STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_WRITELBLK,
     *          IOSB,,,B, %VAL( 8 ),,,,)
C
C
        STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_READLBLK,
     *                          IOSB,,,B, %VAL( 4 ),,,,)
        BUFF(1) = B(1) .AND. 1023
        BUFF(2) = 512 - (B(2) .AND. 1023)
      RETURN
C
C     WRITE TARGET
C
 1000 CONTINUE
        LN   = 512 - BUFF (2)
        ELM  = BUFF(1)
        ELM  = IOR ( ELM, '0800'X )
        LN   = IOR ( LN, '0800'X  )
        PRINT *,' X  AND  Y  ',ELM,'  ',LN
        B(1) = IOR ( SPD, '0080'X )
        B(2) = IOR ( LPR, '0011'X )
        B(3) = IOR ( LPA, '0000'X )
        B(4) = IOR ( LPD, ELM     )
        B(5) = IOR ( LPD, LN      )
        STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_WRITELBLK,
     *          IOSB,,,B, %VAL( 10 ),,,,)
      RETURN
C
C     ERASE IMAGE NO LONGER USED SUBROUTINE CLRGI USED
C     LEFT IN CASE SOMEONE HAS OLDER ROUTINES WHICH DO NOT CALL CLRGI
C
 1100 CONTINUE
        IMG   = ABS(NC)
        IMG   = 2**(IMG - 1)
        PRINT *,' IMG = ',IMG
        B(1) = IOR ( LDC, IMG )
        B(2) = IOR ( LSM, '000F'X )
        B(3) = ERS
        STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_WRITELBLK,
     *          IOSB,,,B, %VAL( 6 ),,,,)
      RETURN
C
C     ERASE GRAPHIC NO LONGER USED SUBROUTINE CLRGI USED
C     LEFT IN CASE SOMEONE HAS OLDER ROUTINES WHICH DO NOT CALL CLRGI
C
 1200 CONTINUE
        GR   = ABS(NC) / 2
        GR   = 2**(GR + 8)
        B(1) = IOR ( LDC, GR )
        B(2) = IOR ( LSM, '0F00'X )
```

```
        B(3) = ERS
        STAT = SYS$QIOW ( , %VAL( GR_LU ),IO$_WRITELBLK,
     *         IOSB,,,B, %VAL( 6 ),,,,)
        RETURN
C
C       ANY OTHER VALUE FOR IOP IS BAD
C
 1300 CONTINUE
        WRITE(6,*) ' ERROR MESSAGE IOP OUT OF RANGE'
        RETURN
        END
C
C
C         CIOPCK IS A SUBROUTINE TO PACK 16 BIT DATA BACK INTO BYTES
C
          SUBROUTINE CIOPCK ( B, BUFF, NUM )
          INTEGER*2 B(512),T2
          BYTE T1(2), BUFF(512)
          EQUIVALENCE ( T2, T1 )
          DO I = 1, NUM
            T2 = B(I)
            IF ( T2.GT.255 ) THEN
              T2 = IAND ( T2, 255 ) + 1
            END IF
            BUFF(I) = T1(1)
          END DO
          RETURN
          END
C
C         CIOGPK IS A SUBROUTINE TO PACK 1 BIT OF EACH BYTE OF AN ARRAY
C         INTO ITS BIT POSITION IN ANOTHER ARRAY.  THIS IS USED TO
C         CONVERT THE RESULT OF A GRAPHIC READ BACK TO THE ELAS FORMAT.
C
          SUBROUTINE CIOGPK ( B, BUFF, NUM )
          BYTE B(512), T1(2)
          INTEGER*2 T2, BUFF(32), MASK(16)
          EQUIVALENCE ( T2,T1 )
          DATA MASK/'8000'X,'4000'X,'2000'X,'1000'X,
     *              '800'X, '400'X, '200'X, '100'X,
     *              '80'X,'40'X,'20'X,'10'X,8,4,2,1/
C         *              '8000'X,'4000'X,'2000'X,'1000'X,
C         *              '800'X, '400'X, '200'X, '100'X/
          LOC = 1
          LOCO = 1
          DO I = 1, NUM, 16
            T2 = 0
            DO J = 1, 16
              IF ( B(LOC).NE.0 ) T2 = IOR ( T2, MASK(J))
              LOC = LOC + 1
            END DO
            BUFF(LOCO) = T2
            LOCO = LOCO + 1
          END DO
          RETURN
          END
```

```
©PY©_
©IMH0150008000©IMV0055010000©ISTF01©IS204©IC000©IT01000©IOP
      SUBROUTINE RDSTAT(XLOC,YLOC,ENTER,FUNC1,FUNC2)
      IMPLICIT INTEGER*2 (A - Z)
      INTEGER*4 SYS$ASSIGN, SYS$QIOW, STAT
      EXTERNAL IO$_WRITELBLK,IO$_READLBLK
      INTEGER*2 GR, IMG, MASK, COL, BUFF(512), B(600), IOSB(4),
     *          XLOC, YLOC,
     *          WID, LSM, WGD, WAC, LWM, LUM,
     *          ERS, ERL, SLU, EGW, LER, LEA,
     *          LDC, NOP, LPR, LPR1, LPR2, LPR3,
     *          SPD, LPA, LPD, RPD, BIT10, BIT11,
     *          BIT15
C
C     THE FOLLOWING PARAMETERS DEFINE THE GRINNEL OPCODES SYMBOLICLY.
C
      PARAMETER
     *(WID='0000'X, LSM='1000'X, WGD='2000'X,
     * WAC='2400'X, LWM='2800'X, LUM='2C00'X,
     * ERS='3000'X, ERL='3400'X, SLU='3800'X,
     * EGW='3C00'X, LER='4000'X, LEA='4800'X,
     * LEB='5000'X, LEC='5800'X, LLR='6000'X,
     * LLA='6800'X, LLB='7000'X, LLC='7800'X,
     * LDC='8000'X, NOP='9000'X, LPR='C000'X,
     * LPR1='C200'X, LPR2='C400'X, LPR3='C600'X,
     * SPD='A000'X, LPA='B000'X, LPD='D000'X, RPD='E000'X,
C
C     BIT MASKS
C
     * BIT10='0400'X, BIT11='0800'X, BIT15='8000'X)
C
         STAT = SYS$ASSIGN ('GRA0:',GR_LU,,)
         B(1) = IOR ( SPD, '0080'X )
         B(2) = IOR ( LPA, '0000'X )
         B(3) = IOR ( RPD, '0000'X )
         STAT = SYS$QIOW(,%VAL(GR_LU),IO$_WRITELBLK,
     *          IOSB,,,B,%VAL(6),,,,)
C
         ENTER = 0
         FUNC1 = 0
         FUNC2 = 0
         B(1) = '0000'X
         B(2) = '0000'X
         DO WHILE (ENTER .EQ. 0)
            STAT = SYS$QIOW(,%VAL(GR_LU),IO$_READLBLK,
     *                      IOSB,,,B,%VAL(16),,,,)
          WRITE (5,10) B(1)
   10     FORMAT( 4X,Z4,/)
          PRINT *,' IOSB =  ',IOSB(1),'  ',IOSB(2)
            IF ( (B(1) .AND. BIT15) .EQ. 0 ) THEN
               ENTER = 0
            ELSE
               ENTER = 1
            END IF
```

```
  END DO
  XLOC = B(1)
  YLOC = B(2)
IF ( ( XLOC .AND. BIT10 ) .EQ. 0 ) THEN
   FUNC1 = 0
ELSE
   FUNC1 = 1
END IF
IF ( ( XLOC .AND. BIT11 ) .EQ. 0 ) THEN
   FUNC2 = 0
ELSE
   FUNC2 = 1
END IF
XLOC = XLOC .AND. 1023
YLOC = YLOC .AND. 1023
  PRINT *,' F1 =  ',FUNC1,'  F2 =  ',FUNC2,'  ENT =  ',ENTER
RETURN
END
```

```
©PY©_
©IMH0150008000©IMV0055010000©ISTF01©IS204©IC000©IT01000©IOP
      SUBROUTINE TRKINT (ENTER)
C
C  POLLS TRACK BALL INTERFACE TO DETERMINE IF THE ENTER BUTTON
C  HAS BEEN PUSHED. THIS REPLACES DISINT FOR THAT FUNCTION
C
       IMPLICIT INTEGER*2 (A - Z)
       INTEGER*4 ENTER
       EXTERNAL IO$_WRITELBLK,IO$_READLBLK
       INTEGER*2 GR, IMG, MASK, COL, BUFF(512), B(600), IOSB(4),
      *          XLOC, YLOC,
      *          WID, LSM, WGD, WAC, LWM, LUM,
      *          ERS, ERL, SLU, EGW, LER, LEA,
      *          LDC, NOP, LPR, LPR1, LPR2, LPR3,
      *          SPD, LPA, LPD, RPD, BIT10, BIT11,
      *          BIT15
C
C     THE FOLLOWING PARAMETERS DEFINE THE GRINNEL OPCODES SYMBOLICLY.
C
       PARAMETER
      *(WID='0000'X, LSM='1000'X, WGD='2000'X,
      * WAC='2400'X, LWM='2800'X, LUM='2C00'X,
      * ERS='3000'X, ERL='3400'X, SLU='3800'X,
      * EGW='3C00'X, LER='4000'X, LEA='4800'X,
      * LEB='5000'X, LEC='5800'X, LLR='6000'X,
      * LLA='6800'X, LLB='7000'X, LLC='7800'X,
      * LDC='8000'X, NOP='9000'X, LPR='C000'X,
      * LPR1='C200'X, LPR2='C400'X, LPR3='C600'X,
      * SPD='A000'X, LPA='B000'X, LPD='D000'X, RPD='E000'X,
C
C     BIT MASKS
C
      * BIT10='0400'X, BIT11='0800'X, BIT15='8000'X)
C
         STAT = SYS$ASSIGN ('GRA0:',GR_LU,,)
         B(1) = IOR ( SPD, '0080'X )
         B(2) = IOR ( LPA, '0000'X )
         B(3) = IOR ( LPR, '0011'X )
         B(4) = IOR ( RPD, '0000'X )
         STAT = SYS$QIOW(,%VAL(GR_LU),IO$_WRITELBLK,
      *          IOSB,,,B,%VAL(8),,,,)
C
         ENTER = 0
C
C         DO WHILE ((ENTER .EQ. 0) .AND. (JUNK .LE. 500))
C
           STAT = SYSSQIOW(,%VAL(GR_LU),IO$_READLBLK,
      *                        IOSB,,,B,%VAL(16),,,,)
C
           WRITE (5,10) B(1)
   10      FORMAT (4X,Z4,/)
           IF ( (B(1) .AND. BIT15) .EQ. 0 ) THEN
              ENTER = 0
```

```
            ELSE
               ENTER = 1
            END IF
C
C        END DO
C
      PRINT *,'ENTER =  ',ENTER
      RETURN
      END
```

```
©PY©-
©IMH0150008000©IMV0055010000©ISTF01©IS204©IC000©IT01000©IOP
      SUBROUTINE ZOOM
      IMPLICIT INTEGER*2 (A - Z)
      INTEGER*4 SYS$ASSIGN,SYS$QIOW,STAT
      EXTERNAL IO$_READLBLK, IO$_WRITELBLK
      INTEGER*2 GR, IMG, MASK, COL, BUFF(512), B(600), IOSB(4),
     *          XLOC, YLOC,
     *          WID, LSM, WGD, WAC, LWM, LUM,
     *          ERS, ERL, SLU, EGW, LER, LEA,
     *          LDC, NOP, LPR, LPR1, LPR2, LPR3,
     *          SPD, LPA, LPD, RPD, BIT10, BIT11,
     *          BIT15
C
C     THE FOLLOWING PARAMETERS DEFINE THE GRINNEL OPCODES SYMBOLICLY.
C
      PARAMETER
     *(WID='0000'X, LSM='1000'X, WGD='2000'X,
     * WAC='2400'X, LWM='2800'X, LUM='2C00'X,
     * ERS='3000'X, ERL='3400'X, SLU='3800'X,
     * EGW='3C00'X, LER='4000'X, LEA='4800'X,
     * LEB='5000'X, LEC='5800'X, LLR='6000'X,
     * LLA='6800'X, LLB='7000'X, LLC='7800'X,
     * LDC='8000'X, NOP='9000'X, LPR='C000'X,
     * LPR1='C200'X, LPR2='C400'X, LPR3='C600'X,
     * SPD='A000'X, LPA='B000'X, LPD='D000'X, RPD='E000'X)
C
       STAT = SYS$ASSIGN ('GRA0:',GR_LU,,)
C
      PRINT *,' TURN FUN A SWITCH OFF'
      PRINT *,'  '
      PRINT *,'  '
      PRINT *,' DEPRESS ENTER TO ZOOM :'
      PRINT *,'  '
      PRINT *,' ZOOM VALUES SEQUENCE THRU MODULO (1, 2, 4, 8) '
      PRINT *,'  '
      PRINT *,'  '
      PRINT *,' TURN FUN A ON THEN DEPRESS ENTER TO QUIT'
      PRINT *,'  '
      PRINT *,'  '
      PRINT *,' MOVE TRACK THEN DEPRESS ENTER TO SCROLL'
      PRINT *,'  '
      PRINT *,'  '
C
      X = 0
      Y = 0
      ENTER = 0
      FUNC1 = 0
      FUNC2 = 0
C
      CALL RDSTAT(X,Y,ENTER,FUNC1,FUNC2)
      FNC2HD = 0
      CHAN = 0
C
```

```
      DO WHILE (FUNC1 .EQ. 0)
C
         ZMVAL = ZMVAL + ENTER
         ZMVAL = MOD (ZMVAL, 4)
         ZOOMV = ZMVAL
         B(1)  = IOR ( SPD, '0100'X   )
         B(2)  = IOR ( LPR, '000F'X   )
         B(3)  = IOR ( SPD, '0008'X   )
         ZOOMV = IOR ( ZOOMV, '004C'X )
         B(4) = IOR ( LPR,  ZOOMV     )
         B(5) = IOR ( LPA, '0000'X    )
         B(6) = IOR ( LPD,    X       )
         B(7) = IOR ( LPD,    Y       )
         INSTCT = 14
C
         IF (FUNC2 .NE. FNC2HD) THEN
            CHAN = CHAN + 1
            CHAN = MOD (CHAN, 3)
            FCHAN = IOR (2**CHAN, '0F00'X)
            FNC2HD = FUNC2
            B(8) = IOR ( LDC, FCHAN   )
            INSTCT = INSTCT + 2
         END IF
C
C          STAT = SYS$QIOW(,%VAL(GR_LU),IO$_WRITELBLK,
C     *                    IOSB,,,B,%VAL(INSTCT),,,,)
C
         CALL RDSTAT(X,Y,ENTER,FUNC1,FUNC2)
C
        END DO
C
C  CENTER IMAGE ON SCREEN WITH ZOOM VALUE OF 1
C
         X = 255
         Y = 255
         B(1)  = IOR ( SPD, '0100'X   )
         B(2)  = IOR ( LPR, '000F'X   )
         B(3)  = IOR ( SPD, '0008'X   )
         B(4) = IOR ( LPR,  '004C'X   )
         B(5) = IOR ( LPA, '0000'X    )
         B(6) = IOR ( LPD,    X       )
         B(7) = IOR ( LPD,    Y       )
         INSTCT = 14
C
         STAT = SYS$QIOW(,%VAL(GR_LU),IO$_WRITELBLK,
     *                   IOSB,,,B,%VAL(INSTCT),,,,)
      RETURN
      END
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NORDA Report 203	5. MONITORING ORGANIZATION REPORT NUMBER(S) NORDA Report 203
6. NAME OF PERFORMING ORGANIZATION Naval Ocean Research and Development Activity	7a. NAME OF MONITORING ORGANIZATION Naval Ocean Research and Development Activity
6c ADDRESS (City, State, and ZIP Code) Ocean Science Directorate NSTL, Mississippi 39529-5004	7b. ADDRESS (City, State, and ZIP Code) Ocean Science Directorate NSTL, Mississippi 39529-5004
8a NAME OF FUNDING/SPONSORING ORGANIZATION Naval Ocean Research and Development Activity 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code) Ocean Science Directorate NSTL, Mississippi 39529-5004	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 63704N PROJECT NO. R1987 TASK NO. 300 WORK UNIT NO. 23508B

11 TITLE (Include Security Classification)
Implementation Issues for Level 0 Image Processor Software within an Application Package

12 PERSONAL AUTHOR(S)
James E. Lennox

13a TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	From _____ To _____	February 1988	30

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR	image processing, software, hardware, ELAS, ERL, PAL, Landsat, Thematic Mapper data, VAX 11/780, VAX 11/750 modules, callable subroutines, COMD, pixel

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Image processing is rapidly emerging as a field with many interesting areas for the computer scientist. Packages are available that allow images to be manipulated in an interactive environment by applying functions to the image. Functions are defined as transformations from the image memory domain to the display domain, which may permanently alter image memory values. This paper describes minimum hardware constraints and the scope of modifications necessary to implement different image processors within the Earth Resources Laboratory Applications Software (ELAS) software environment. It also contrasts the impact on software engineering principles related to the implementation of low-level software to support the imaging functions.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
UNCLASSIFIED/UNLIMITED SAME AS RPT ■ DTIC USERS	Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL James E. Lennox	22b TELEPHONE NUMBER (Include Area Code) (601) 688-4633	22c OFFICE SYMBOL Code 351

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE